THE UNIVAC 1050 FASTRAND SOFTWARE SYSTEM

E. Davies
Systems Programming - Blue Bell
March, 1966

1.0 Introduction

2.0 General

3.0 System Introduction

- 3.1 Desk Mapping of Fastrand
- 3.2 Processing of Fastrand Software Package
- 3.3 IDMS
- 3.4 SADL
- 3.5 AJAXF and OPUSF
- 3.6 Operating System

4.0 Using the System

- 4.1 DLL Loading Sequence
- 4.2 Assembly of Programs with PAL Drum
- 4.3 Load of Object Programs
- 4.4 Maintenance of Source and Object Code Programs

5.0 Files

- 5.1 Basic and Item Level Routines
- 5.2 File Design Consideration

6.0 System Maintenance and Protection

- 6.1 Recovery Dump and Load
- 6.2 Remapping the Drum

7.0 Conclusion

The UNIVAC 1050 Fastrand Software System

1.0 INTRODUCTION

These notes cover the subject of the 1050 Fastrand Software System from the point of view of its design concepts, its introduction to an installation, and operational use.

Detailed operating instructions are not, in general, described, this being the function of the documentation of the software in question.

2.0 GENERAL

The Fastrand software is similar externally to that for tape or card systems. Changes have been made where necessary or desirable to take advantage of the mass-storage device. Notably the assembly times of symbolic programs is cut significantly.. Certain of the standard software is new, being applicable to this system only.

From the users point of view, a mass-storage device can appear somewhat monolithic. Problems arise such as where to put data, how to update files, how to protect data from accidental destruction, addressing techniques, recovery procedures, and others.

The software in question solves these problems in a general purpose way, while at the same time care has been taken to allow users to extend the available programs and still retain the flexibility and protection of the standard software.

In addition, hybrid systems could be built having tape software for some parts of an application and drum software for other parts.

3.0 SYSTEM INTRODUCTION

To set up a complete software system on the Fastrand, the following sequence should be followed:

3.1 Desk mapping of Fastrand.

An installation may have one or more Fastrands, each capable of containing huge amounts of data. Mapping the Fastrand(s) means to allocate certain areas of contiguous sectors for certain purposes. Such an area of sectors is called a CLASS. Any number of classes may be defined after which the user may think of himself as having several classes of mass-storage rather than one or more Fastrands.

Classes should be defined to cater for

a) the standard software
b) the users programs
c) data file areas.

A preliminary definition of classes could take the following tabular form:

Class Type	Size of Directory	Size of Class	
*SCRACH	n_1 = 1 sector	m_1 sectors	*SCRACH area required for AJAX, PAL and FRDP.
**MASTER	n_2 = 1 sector	m_2 sectors	**MASTER present for equivalent of standard system tape to exist on drum.
*A---- thru *Z----	n_3 = 1 sector	m_3 sectors	Up to 26 program classes for particular programmers or working groups to keep programs.
AA----	n_4 sectors	m_4 sectors	Any number of data files having any 8 character a/n name starting with an alphabetic character. These may be assigned at the users discretion to reflect different jobs, working groups or other criteria.

The values of n and m should be calculated at this stage. $n_1 = n_2 = n_3 = 1$. This indicates a dummy FD generated in an * area for control purposes. n_4 = maximum number of data files envisaged in that particular class.

m_1 is the scratch area size which may be determined by the requirements of either AJAXF, PAL Drum, or FRDP.

a) AJAXF m_1 = maximum size of an *area = m_2 or m_3 maximum

b) PAL Drum n_1 = 2 times maximum number of lines to be printed during one assembly

c) FRDP $m_1 = x + y + 1$ where x = largest value of n_4
y = largest number of FILE control cards in one run.

In practice, the AJAXF requirement will determine the scratch size m_1.

m_2 and m_3 can be calculated knowing that 1 sector will contain 2 source code tape blocks or cards, or 1 object code tape block or card. Thus, for example, a library tape containing 10000 blocks of source code will require 5000 sectors.

m_2 and m_3 should also allow generously for expansion of the program library and the way in which AJAXF and OPUSF update programs. (See Section 4.4 on AJAXF, OPUSF.)

m_4 can only be decided by the user who must know roughly the size of his projected data files.

The above desk mapping puts one in a position to actually map the drum using the program IDMS. Planning, preparation and thought at this stage will save much later trouble due to faulty space estimates. If in doubt always over-estimate requirements, especially in the early stages of introduction of the system.

3.2 Processing of Fastrand Software Package

The Fastrand Software is available on a system tape or as a card deck. Card decks will be distributed when a machine configuration does not include tapes (IIIC, IIIA, or VIC).

Within the system exist certain programs needed to initialize the drum and a set of card program loaders for any card reader.

If the user receives a tape he should punch, using the OPUS provided on that tape, the card loader appropriate to him (for example, LD8X for 80 col. serial reader), IDMS, and DLL.

Remove the first card from DLL.

Place these cards in the sequence loader, IDMS, DLL.

3.3 IDMS

The Initial Drum Setup routine (IDMS) will actually map the drum to reflect the allocation of classes defined under Section 3.1.

Referring to the IDMS description and the desk mapping, IDMS directive cards can now be keypunched and should be placed after DLL in the card deck produced as described in the previous section, (3.2).

The resulting card deck now constitutes a complete IDMS run. IDMS then writes DLL (Drum Locator Loader) on to the drum in the first few sectors, and then constructs the System Directory (SD) from the IDMS directive cards.

This IDMS run should now be initiated through the LOAD CARD console control. Displays and printouts will be as described in the IDMS write-up.

At this point the Fastrand will contain DLL, the SD, and an FDD (File Directory Descriptor) for each class defined.

3.4 SADL

This section applies to recipients of a Fastrand system tape rather than a card deck. If the user wants the equivalent of a master systems tape on the drum this should now be done using SADL to write the appropriate parts of the system tape to the ** SYSTEMS area.

Call SADL from the systems tape using the normal $SADL call card. The following control cards will be as described in the SADL documentation. For example, to transfer the entire tape to the **SYSTEMS area, the control cards would be

```
INOUT  **
COPY
STOP
```

SADL may also be used at any time to transfer any programs on tape, object or source, to any *area.

3.5 AJAXF and OPUSF

AJAXF and OPUSF are program maintenance programs analogous to their tape counterparts.

At this point of the introduction of the system they should be used by recipients of the Fastrand software card deck to set up the ##MASTR area on drum. This is the same function as that performed by SADL on tape in Section 3.4.

Note that AJAXF accepts source code for transfer to drum, and OPUS accepts object code. Object decks of AJAXF and OPUSF are part of the deck supplied, and each should be preceded by the appropriate loader, also supplied, to enable the program to load.

3.6 Operating System

Thus far in setting up the system there has been no need to use a Fastrand version of the Operating System (OPR). The programs used, OPUS, IDMS, SADL, AJAXF have all been loaded via the appropriate card loader or from tape.

However, the user will probably want an OPR with which to run his own programs in a drum environment.

The **MASTER area now contains the source code version of OPR if sections 3.4 or 3.5 were followed.

This should now be assembled using the drum assembler which should also now be in **MASTER.

The following sample card deck should be prepared for this drum assembly

```
$DPAL*
**
     OPRF  BEGIN  0520
           OPR    p1.... pn
           END    START
```

Choose OPR p_6 = F or FL.

F means that a Fastrand handler is contained, FL also means that relocatable or absolute programs may be loaded by OPR from the drum. FL will be wanted by users of the full system.

To operate the drum assembly take the following steps:

a) Set the inhibit tape load switch in the tape synchronizer, and reset the inhibit load switch in the Fastrand synchronizer cabinet.

b) Press LOAD TAPE
START — Reads Sector ∅
CONTINUOUS
START — Loads DLL
START — DLL reads in DPAL which then takes over.

The output of DPAL, in this case the object OPR, is located in the Scratch area *SCRACH.

An AJAXF run should now transfer the object program to a more permanent location, the **MASTER area or maybe another *area.

Such an AJAXF run will have the following cards to transfer OPRF to the **MASTER area from the scratch area:

```
$AJAXF*
      INOUT    **
      FILE
      STOP
```

AJAXF exists in the **MASTER area and is called in the same way as DPAL, described above.

4.0 USING THE SYSTEM

At this point, if Section 3.0 was followed, the **MASTER tape contains the Fastrand software necessary to assemble programs and run them using the Fastrand as the main storage medium for source programs, object programs, and data files.

4.1 DLL Loading Sequence

The following standard programs are absolute, run without OPRF control and are loaded by DLL:

DPAL, AJAXF, OPUSF, SADL, FRDP, FRLD

The loading sequence for these is as described in Section 3.6. This loading sequence applies even if OPRF exists in memory. OPRF cannot load programs from the drum which overlay OPRF.

In addition, unlike the tape system, OPRF is itself loaded by DLL using a call card $OPRF* to get it from the **MASTER area.

4.2 Assembly of Programs with PAL Drum

Section 3.6 gave instructions for the assembly of OPRF. The same applies to any other program. The $DPAL* card is followed by one or more cards specifying the areas from which procedures may be called. The object program is placed in the *$SCRACH area from which it may be read into memory directly, punched out using OPUSF, or transferred to another * area by AJAXF.

4.3 Load of Object Programs

As described in Section 4.1, all absolute programs which would overlay OPR must be loaded by DLL.

All other object programs must be loaded by OPRF using a call card of the form $XXXXY where XXXX is the program ID and Y is the systems area identifier of the class where the program is located on the drum. In addition, Y may of course be blank or a digit to specify loading from tape.

4.4 Maintenance of Source and Object Code Programs

AJAXF and OPUSF maintain programs stored in * areas on the drum. The facilities of those utility programs is roughly the same as for tape AJAX and OPUS. Programs may be filed, copied, deleted, corrected, etc.

The user should note that a correction or replacement affecting a program will result in the new version of that program being written elsewhere in the * area. The original program will still be there but inaccessible in the normal way. Any program is located by its program directory (PD) which has been deleted for the old version of the program and reconstructed for the new.

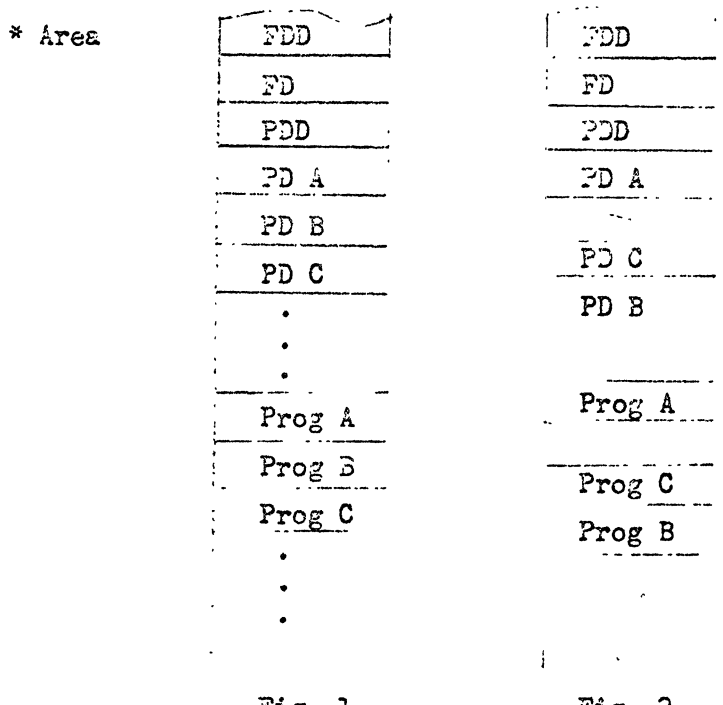


Fig. 1 Fig. 2

If the area drawn in Figure 1 contains 3 programs A, B, C, then to replace Prog. B would result in Figure 2.

In allocating * areas by IDMS, it is best to allow for plenty of redundant space. Such redundant space is recoverable by an AJAXF copy.command which relocates all programs, supressing those now redundant.

Many will find it useful to keep two * areas (in the same manner as 2 system tapes), one being the previous generation of the other. This should also be taken into account in planning the IDMS run. There is no need, however, as with the tape system, to keep a short systems area to speed up the assembler.

5.0 FILES

Data files may be written with the aid of either the Basic Level or the Item Level routines

5.1 The Basic Level Routines

Roughly, the Basic Level operation will cause a file area to be allocated under the control of an FD entry. After allocation of the file area, the user may perform any Fastrand function within that area. Various basic I/O commands are provided for this purpose.

At the Item level an item is assumed to occupy 1 sector, and files are built up allowing multi-item records to be automatically chained.

Thus a record may look symbolically as in Figure 3.

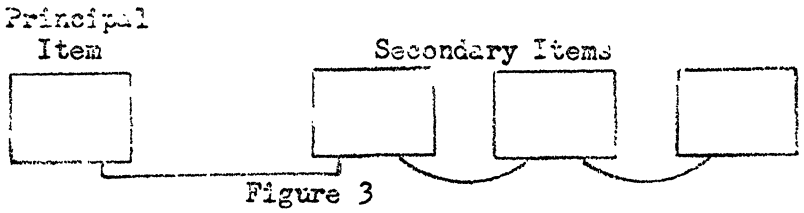


Figure 3

The first sector would be accessed directly and the others by following the chain.

The relationship between the directory and an item level data file within a class is shown in Figure 4.

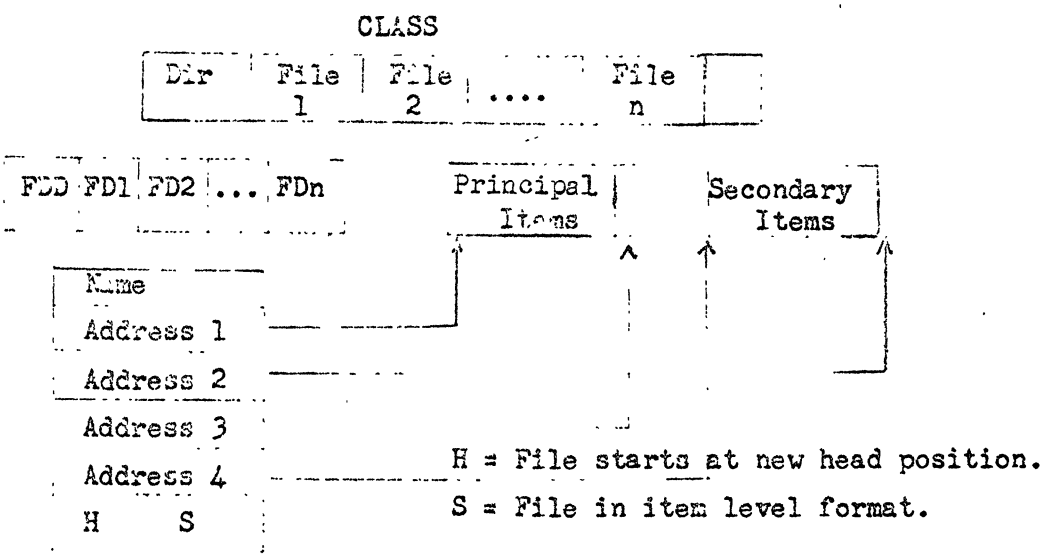


Figure 4

A basic-level file has a similar relationship, except that S is not set. Address 3 and Address 4 are preset to equal Address 1 and Address 2-1 respectively, but will not change as a result of basic level software.

5.2 File Design Consideration

With tape or card files the user has little choice but to access records in his files sequentially.

As is well known, this is not the case with random-access devices such as the Fastrand.

Another important characteristic which influences file processing is that records are updated in place with no change or access to those records immediately preceding or succeeding.

Results of such characteristics are that file updating time is proportional to activity and not to file size. A user with large but low-activity files would, therefore, profit considerably. Also the need to sort files decreases significantly with random-access capabilities, though not entirely.

File record sizes tend to be oriented to a convenient hardware unit, the sector in the case of the Fastrand, and larger records achieved by grouping 2 or more sectors together in a chain.

Initiation of a file requires a decision as to how access to the file is to be made.

Such access can be:

a) Sequential as with tape files. This is useful for very small files, directions and indexes. However, adding new records will cause a sequence problem unless the entire file is rewritten, while sorting of the file will be often needed as in the case of a tape file.

b) by hardware search for match of a key. This method is satisfactory provided the file size or potential search area is such that the search time is not too great. A file written in a sequential mode could well be read by searching.

c) direct by a relative address computed from some key. This approach requires more thought but is more flexible. Sorting of files is minimized, adding of new records is no problem, and a control is possible over average chain length (access time).

   In this mode of operation the decision then has to be made how to randomize the address of a record. This may be done in many ways, for example, a part number 123 might be taken to imply the 'random' address 123. Many keys, however, are alphanumeric and more complex and do not lend themselves

to this solution. Another approach is to form a binary hash total of the record key. This hash total would then be the random address. Such a random number could be cut off so that a maximum is never exceeded. This maximum should be related to the overall file size and this enables the user to predict an average chain length of records. Thus, if the maximum random address is small compared with the number of items, then long chains will accumulate where otherwise independent items generated the same random address. In such a system, however, short chains will imply wasted sectors where particular random numbers did not occur. The user must then balance poor access times of chains against waste of drum space for a particular file.

Figure 5 shows the relationship of % free principal items, and average chain lengths against the ratio of the maximum random number to the total number of items written.

If this ratio is called the chain factor then, if greater than 5, few principal item sectors are wasted. At this point and beyond however average chain length tend toward the chain factor.

On the other hand if the chain factor is 1, we have average chain lengths of about 1.6 but 40% wasted principal sectors.

To optimize access further, a file might be split into 'subfiles' each reflecting different levels of activity, or each placed within one head position to eliminate head movement.

Optimization to allow for hardware characteristics should, however, be used with discretion since future changes of equipment may make such efforts redundant and less compatible.

6.0 SYSTEM MAINTENANCE AND PROTECTION

The Fastrand has the characteristic that many data files are potentially accessible at all times.

If one thinks of a Fastrand file as being equivalent to a file on an unprotected tape which is left permanently mounted, it is obvious that procedures are necessary to avoid accidents by programs and operators, and to recover when such accidents do occur.

Hardware lockout is optionally available to Fastrand users. Lockout is from sector Ø upwards where the upper limit is set manually to one of a number of positions.

In addition software provides elaborate protection to users of the system, and this should normally suffice. However, no software can stop destructive writing on the drum in the last resort. It is, therefore, necessary to plan for and execute scheduled recovery dumps of the drum to provide complete protection.

Dumps of this nature are normally to tape, but card output is feasible for small quantities of data. The program to do this is FRDP, while FRLD will reload the data dumped by DRDP.

6.1 Recovery Dump and Load

FRDP is oriented to dumping part or all of a single class, and can handle several classes by stacking the input control cards. Because the dumping of a Fastrand to tape can take a fair amount of time, it is wise to dump selectively where possible. For example, scratch class areas should not normally be saved in a routine dump. Also classes may contain data files which are updated only infrequently. In this case, a dump taken after each update enables its reconstruction in the event of error.

On a daily basis scheduled dumps could be made to precede preventative maintenance and program testing, followed by a subsequent relaod.

It should be noted that recovery dumps of the type which are normal when updating tape files are not possible with mass-storage. This type assumes an unchanged input file (or files) so that skipping back a few blocks to a special sentinel and environment dump is relatively easy. With mass-storage files, however, updating is normally done 'in place' so that the input files are also the output files and are constantly changing. Recovery must, therefore, be either immediate, or result from loading a previous precautionary dump of the file in question and starting the job again.

It should be noted that in addition to scheduled dumps, any class may be dumped at any time and subsequently reloaded without interfering with other classes.

To reduce the dumping and loading problem and make it more manageable it may be as well to arbitrarily limit the number of classes rather than allow it to grow out of proportion.

In a preventative dump of the entire active system, the first few sectors should also be saved. These contain the program DLL and the Systems Directory (SD). This area is outside any class and must, therefore, be dumped in an absolute mode.

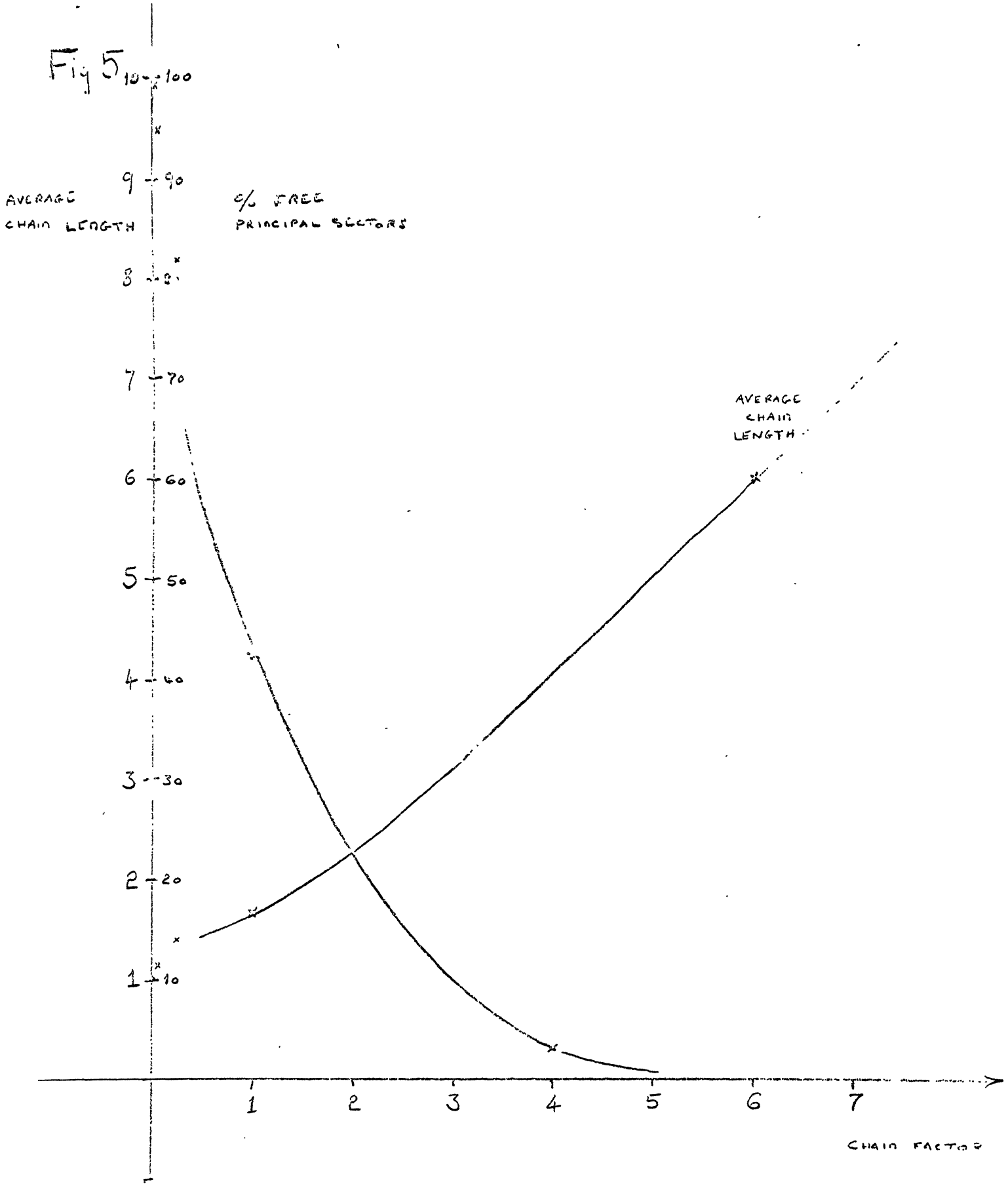
Also classes which exist but contain no valid data should be dumped if there is a possibility that the FDD of that class is destroyed. To dump the FDD only the command

FILE FLEDIRCT, FLEDIRCT is used.

The following is an example of preventative dump FRDP and FRLD control decks, dumping classes AA----AA, BB----BB, CC----CC where class CC----CC contains no active data.

```
$FRD3*                          Prog. call from *area

CLASS

FDUMP  ,P2                      Dump DLL, and SD.
                                P2 = A2 of SDD.

CLASS AA----AA

FILE

CLASS CC----CC

FILE FLEDIRCT, FLEDIRCT

STOP

$FRL3*                          Load FRLD from *area,
                                assuming still valid

LOAD                            LOAD DLL and SD

LOAD AA----AA

LOAD BB----BB

LOAD CC----CC

STOP
```

Fig 5


AVERAGE CHAIN LENGTH
% FREE PRINCIPAL SECTORS
AVERAGE CHAIN LENGTH
CHAIN FACTOR
10
9
8
7
6
5
4
3
2
1
100
90
80
70
60
50
40
30
20
10
1
2
3
4
5
6
7

6.2 Remapping the Drum

Mapping the drum with IDMS is a basic step which should have been made after profound thought for the requirements of the system.

To avoid wildly complex software it is decreed not possible to dynamically remap the drum.

It is necessary instead to dump the drum (those classes requiring saving), run IDMS with new mapping, and then reload the classes previously dumped. FRLD takes care of any change in the SD made between dumping and reloading so that all data is reloaded class-relative. If classes AA----AA, BB----BB, CC----CC are to be saved in such a run, the FRDP and FRLD control decks are as follows:

```
$FRD3*                          Call of FRDP from *area
                                on drum.

CLASS AA----AA

FILE

CLASS BB----BB

FILE

CLASS CC----CC

FILE

STOP

$FRL1                           Call FRLD from tape
                                if *area location changed
                                by IDMS.

LOAD AA----AA

LOAD BB----BB

LOAD CC----CC

STOP
```

No attempt should be made to save DLL or the SD which are re-written by IDMS under these circumstances.

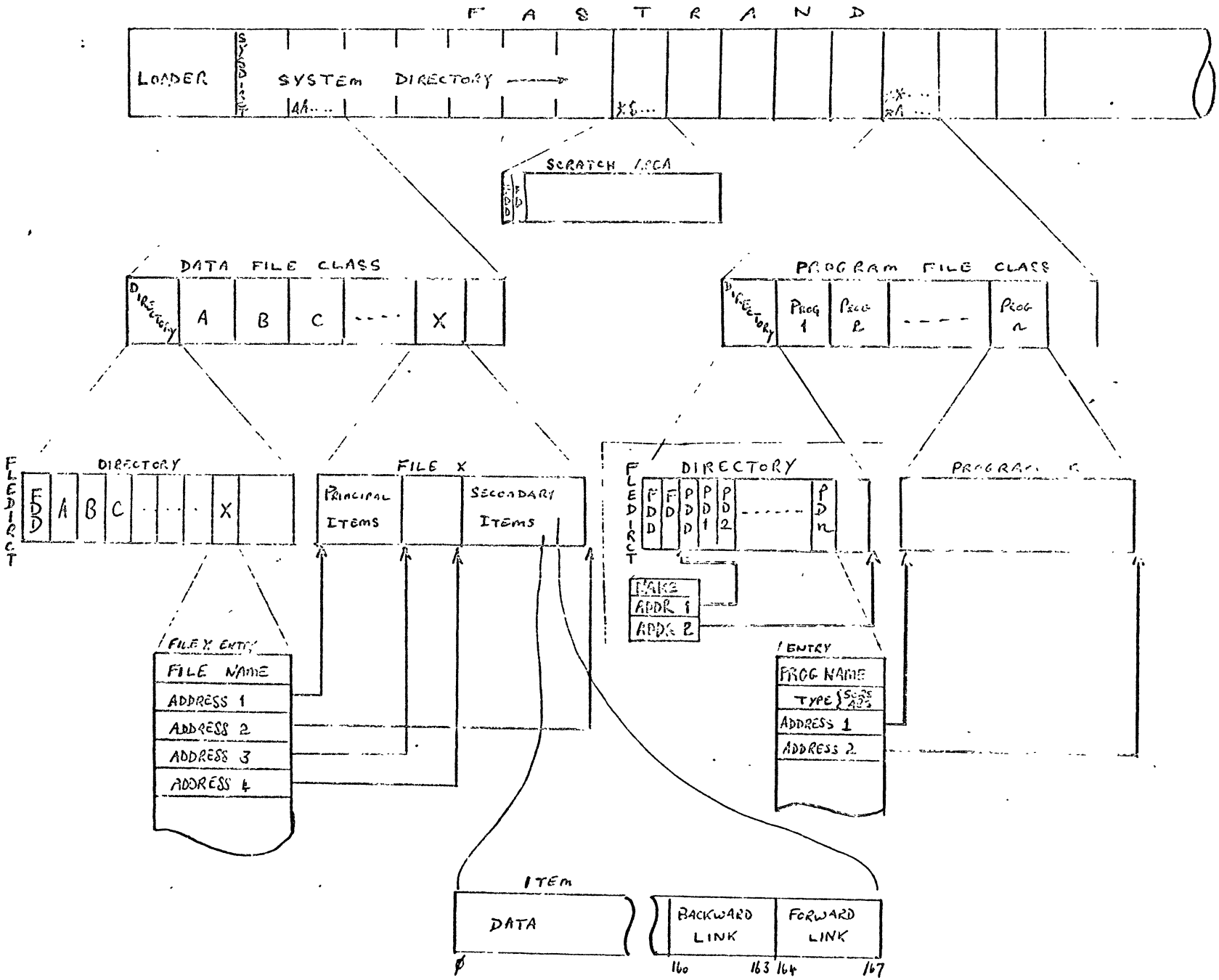

FASTRAND
LOADER
SYSTEM DIRECTORY
SCRATCH AREA
DATA FILE CLASS
DIRECTORY
A
B
C
X
PROGRAM FILE CLASS
PROG 1
PROG 2
PROG n
FLEDIRECT
FDD
PDD1
PDD2
PDDn
FILE X
PRINCIPAL ITEMS
SECONDARY ITEMS
NAME
ADDR 1
ADDR 2
FILE X ENTRY
FILE NAME
ADDRESS 1
ADDRESS 2
ADDRESS 3
ADDRESS 4
ENTRY
PROG NAME
TYPE
ADDRESS 1
ADDRESS 2
ITEM
DATA
BACKWARD LINK
FORWARD LINK
0
160
163
164
167

FIGURE 6

7.0 CONCLUSION

The schematic in Figure 6 shows the entire system of classes and directives and their interrelationship.

To the usual advantages of mass-storage equipment may be added the benefit arising from the enforced planning, preparation, and machine discipline imposed upon the user by the system.

The price of increased flexibility is paradoxically increased control, without which the interaction of the various parts of the software could initiate chaos.